(12) UK Patent Application (19) GB (11) 2 322 994 (13) A

(43) Date of A Publication 09.09.1998

(21) Application No 9804913.3

(22) Date of Filing 06.03.1998

(30) Priority Data
(31) 9704630 (32) 06.03.1997 (33) GB

(71) Applicant(s)
LSI Logic Corporation
(Incorporated in USA - California)
1551 McCarthy Boulevard, MS-D-106, Milpitas, California 95035, United States of America

(72) Inventor(s)
Simon Bewick

(74) Agent and/or Address for Service
Miller Sturt Kenyon
9 John Street, LONDON, WC1N 2ES, United Kingdom

(51) INT CL^6
H04N 7/167

(52) UK CL (Edition P)
H4F FDE FD12X FD30K

(56) Documents Cited
EP 0805599 A2
Prior art acknowledged on pages 1 and 2 and figure 2 of the application

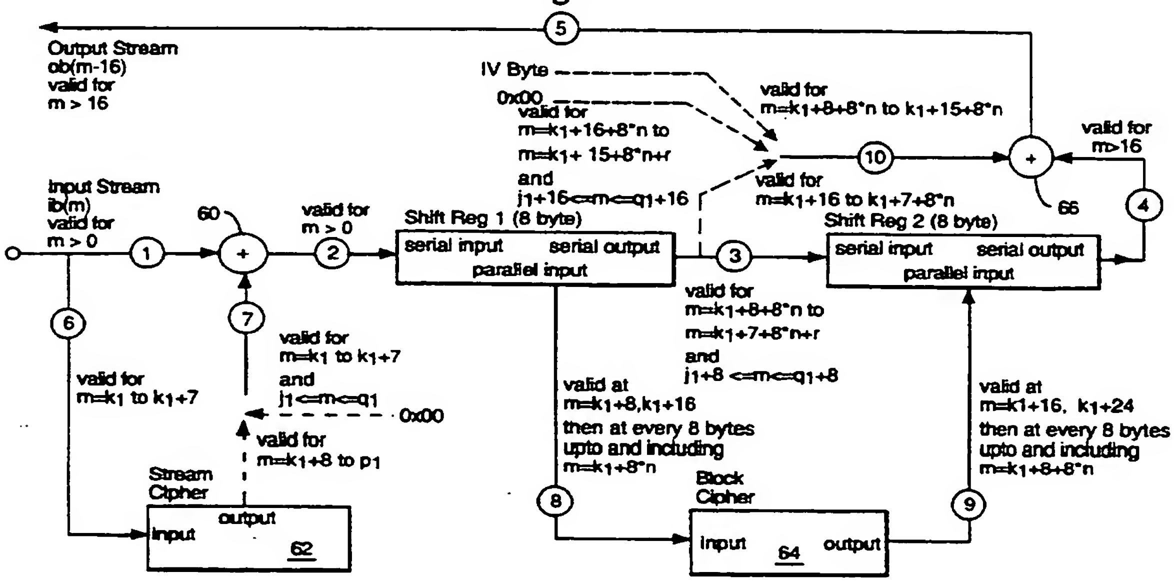
(58) Field of Search
UK CL (Edition P) H4F FDE
INT CL^6 H04N 7/16 7/167
Online: WPI, INSPEC

(54) Abstract Title

Descrambling DVB data according to ETSI common scrambling specification

(57) In order to descramble sections of scrambled data interleaved with sections of unscrambled data in a transport stream of broadcast video data, while leaving the sections with the original timing relationship in the transport stream, a common data flow path is provided both for sections of scrambled data and sections of unscrambled data. Signal path loops 6,7; 8,9 include cipher means 62,64 to enable the descrambling of scrambled data, and a control state machine controls the flow of data through said common data flow path and said signal path loops to enable passage of unscrambled data sections and descrambling of scrambled data sections, while maintaining the desired relative positions of the data sections.

Fig.6.



GB 2 322 994 A

At least one drawing originally filed was informal and the print reproduced here is taken from a later filed formal copy.

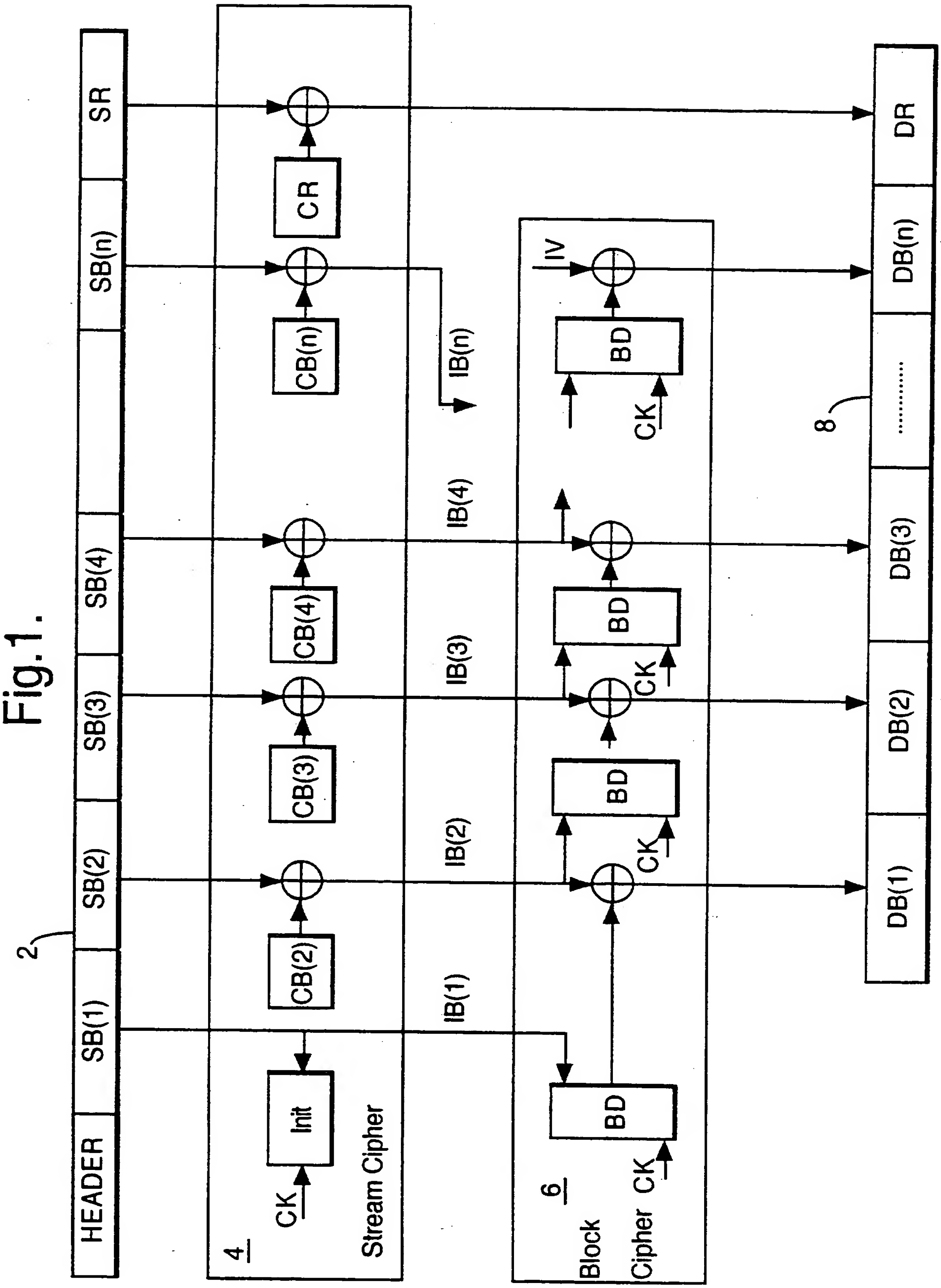

HEADER
SB(1)
SB(2)
SB(3)
SB(4)
SB(n)
SR
2
4
Stream Cipher
CK
Init
CB(2)
CB(3)
CB(4)
CB(n)
CR
IB(1)
IB(2)
IB(3)
IB(4)
IB(n)
6
Block
Cipher
CK
BD
BD
BD
BD
IV
8
DB(1)
DB(2)
DB(3)
DB(n)
DR

Fig.1.

Fig.2.

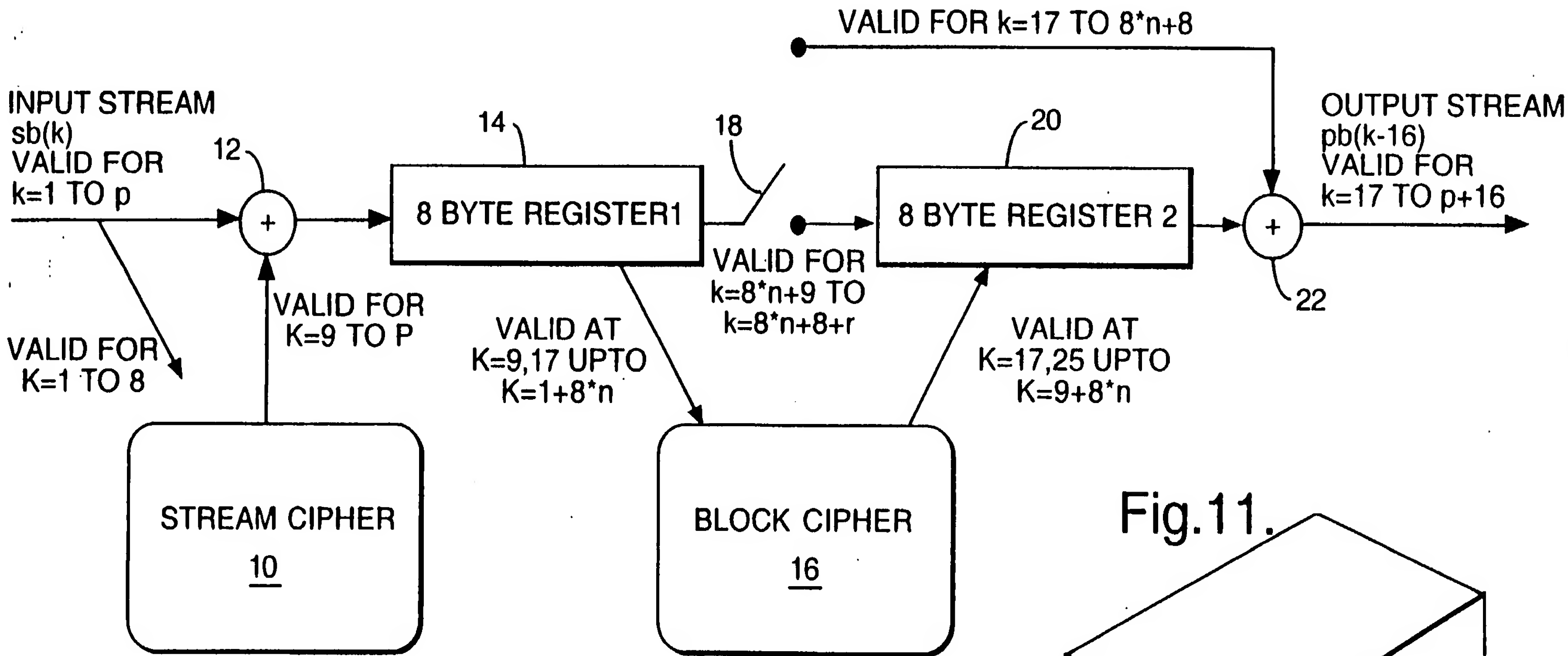

VALID FOR k=17 TO 8*n+8
INPUT STREAM
sb(k)
VALID FOR
k=1 TO p
12
14
8 BYTE REGISTER1
18
20
8 BYTE REGISTER 2
OUTPUT STREAM
pb(k-16)
VALID FOR
k=17 TO p+16
22
VALID FOR
K=1 TO 8
VALID FOR
K=9 TO P
VALID AT
K=9,17 UPTO
K=1+8*n
VALID FOR
k=8*n+9 TO
k=8*n+8+r
VALID AT
K=17,25 UPTO
K=9+8*n
STREAM CIPHER
10
BLOCK CIPHER
16

Fig.11.

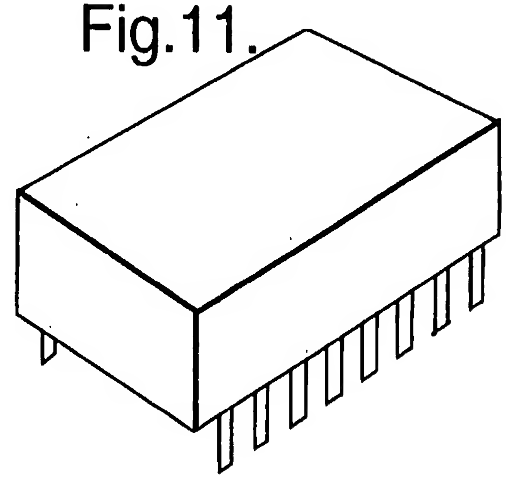
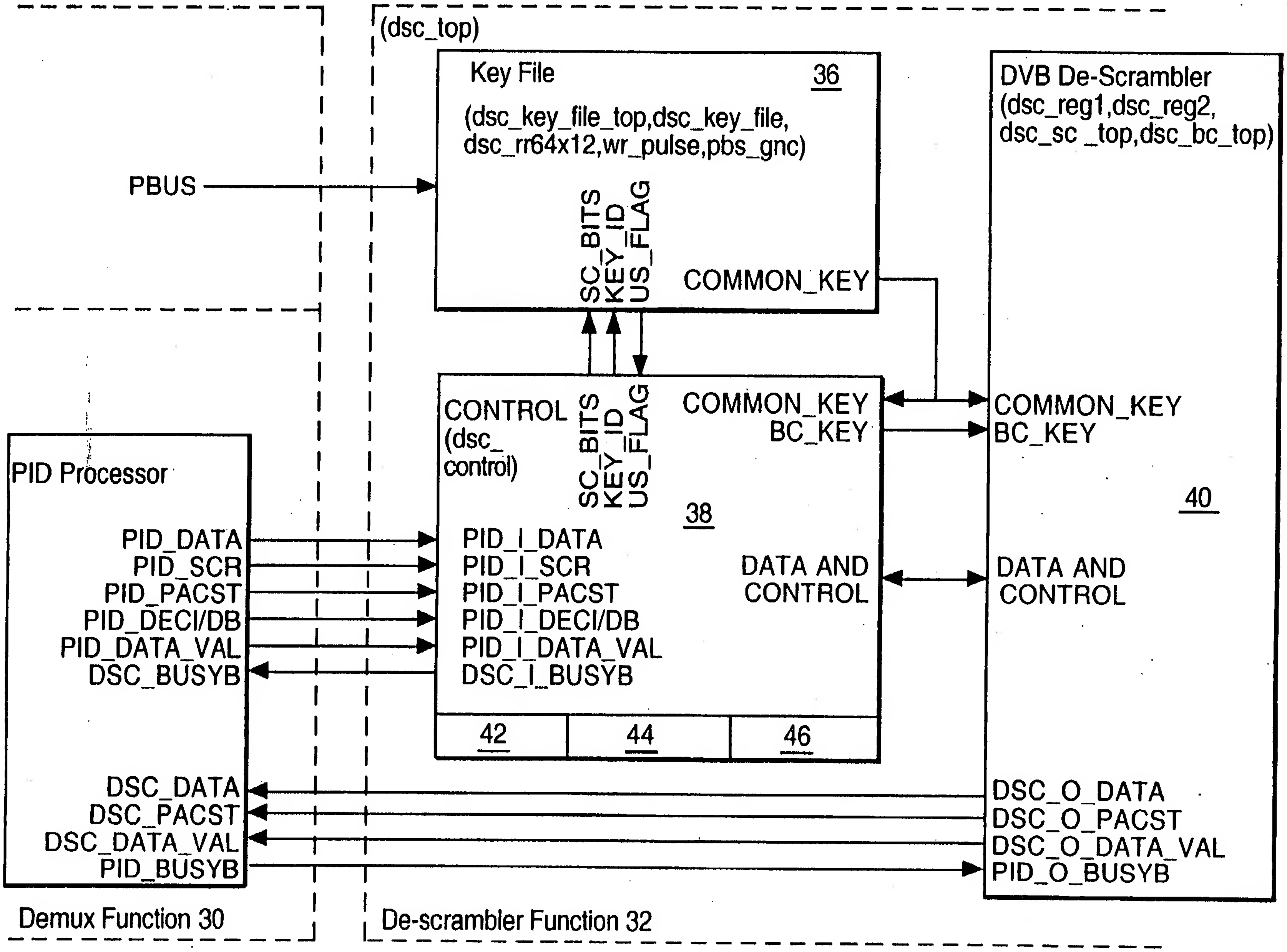
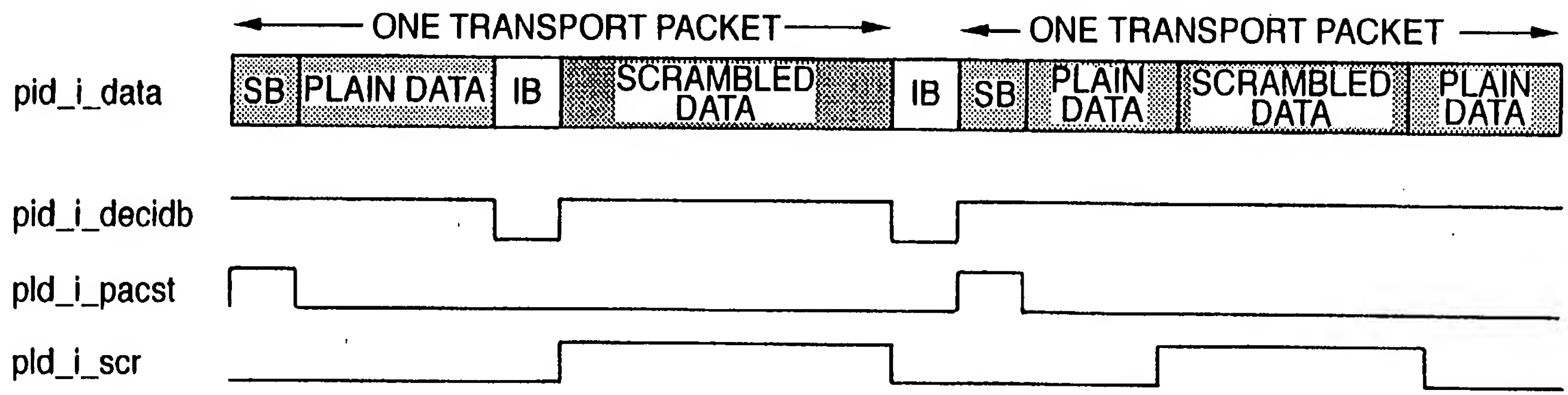


Fig.3.



(dsc_top)
Key File
36
(dsc_key_file_top,dsc_key_file,
dsc_rr64x12,wr_pulse,pbs_gnc)
PBUS
SC_BITS
KEY_ID
US_FLAG
COMMON_KEY
DVB De-Scrambler
(dsc_reg1,dsc_reg2,
dsc_sc _top,dsc_bc_top)
40
CONTROL
(dsc_
control)
38
COMMON_KEY
BC_KEY
DATA AND
CONTROL
PID_I_DATA
PID_I_SCR
PID_I_PACST
PID_I_DECI/DB
PID_I_DATA_VAL
DSC_I_BUSYB
42
44
46
PID Processor
PID_DATA
PID_SCR
PID_PACST
PID_DECI/DB
PID_DATA_VAL
DSC_BUSYB
DSC_DATA
DSC_PACST
DSC_DATA_VAL
PID_BUSYB
DSC_O_DATA
DSC_O_PACST
DSC_O_DATA_VAL
PID_O_BUSYB
Demux Function 30
De-scrambler Function 32

Fig.4.


ONE TRANSPORT PACKET
ONE TRANSPORT PACKET
pid_i_data
SB
PLAIN DATA
IB
SCRAMBLED DATA
IB
SB
PLAIN DATA
SCRAMBLED DATA
PLAIN DATA
pid_i_decidb
pld_i_pacst
pld_i_scr

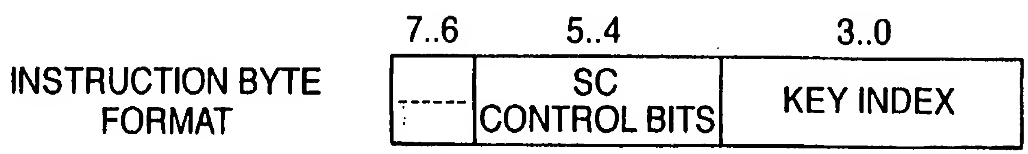

INSTRUCTION BYTE FORMAT
7..6
5..4
3..0
SC CONTROL BITS
KEY INDEX

Fig.5.

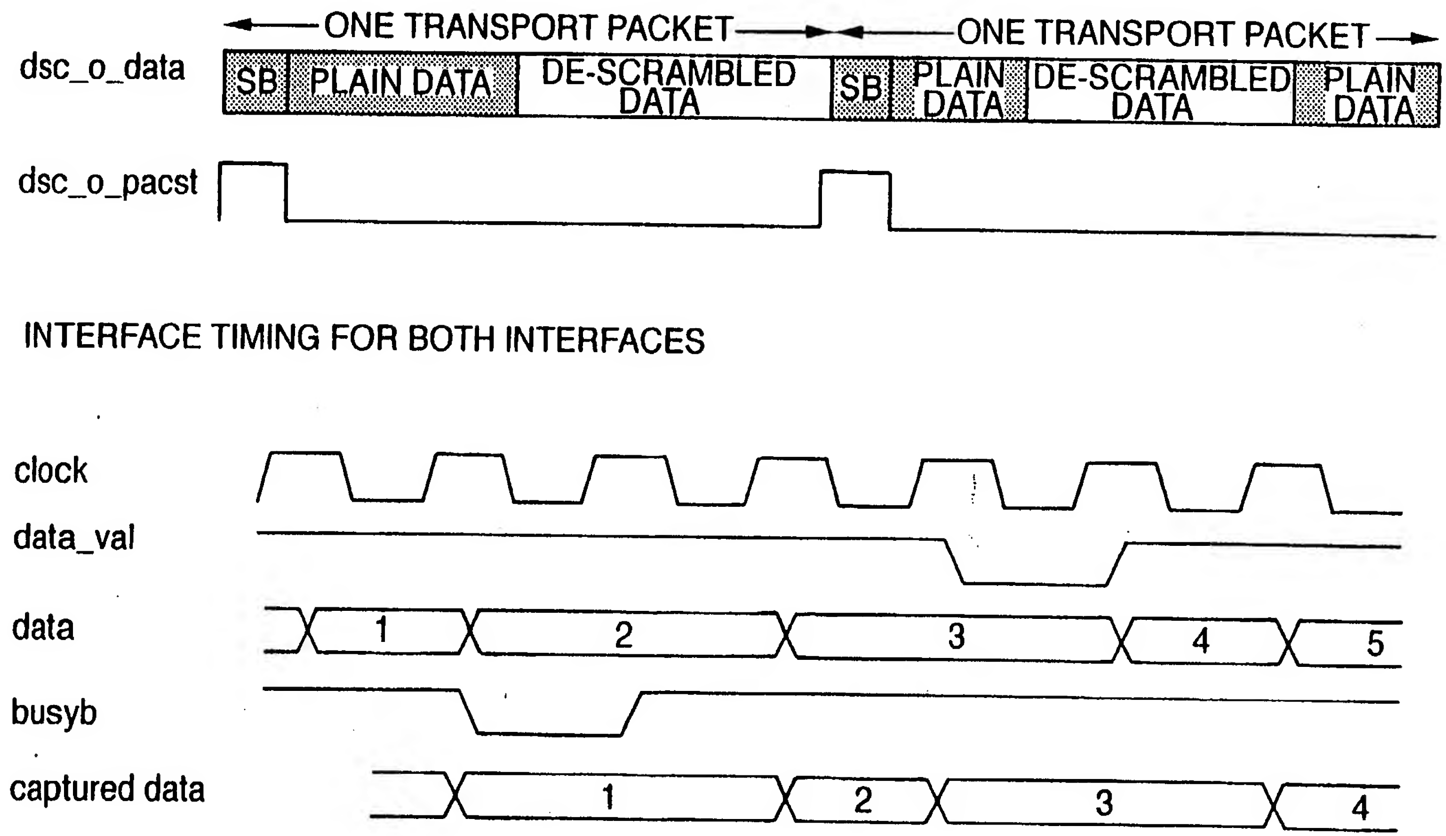
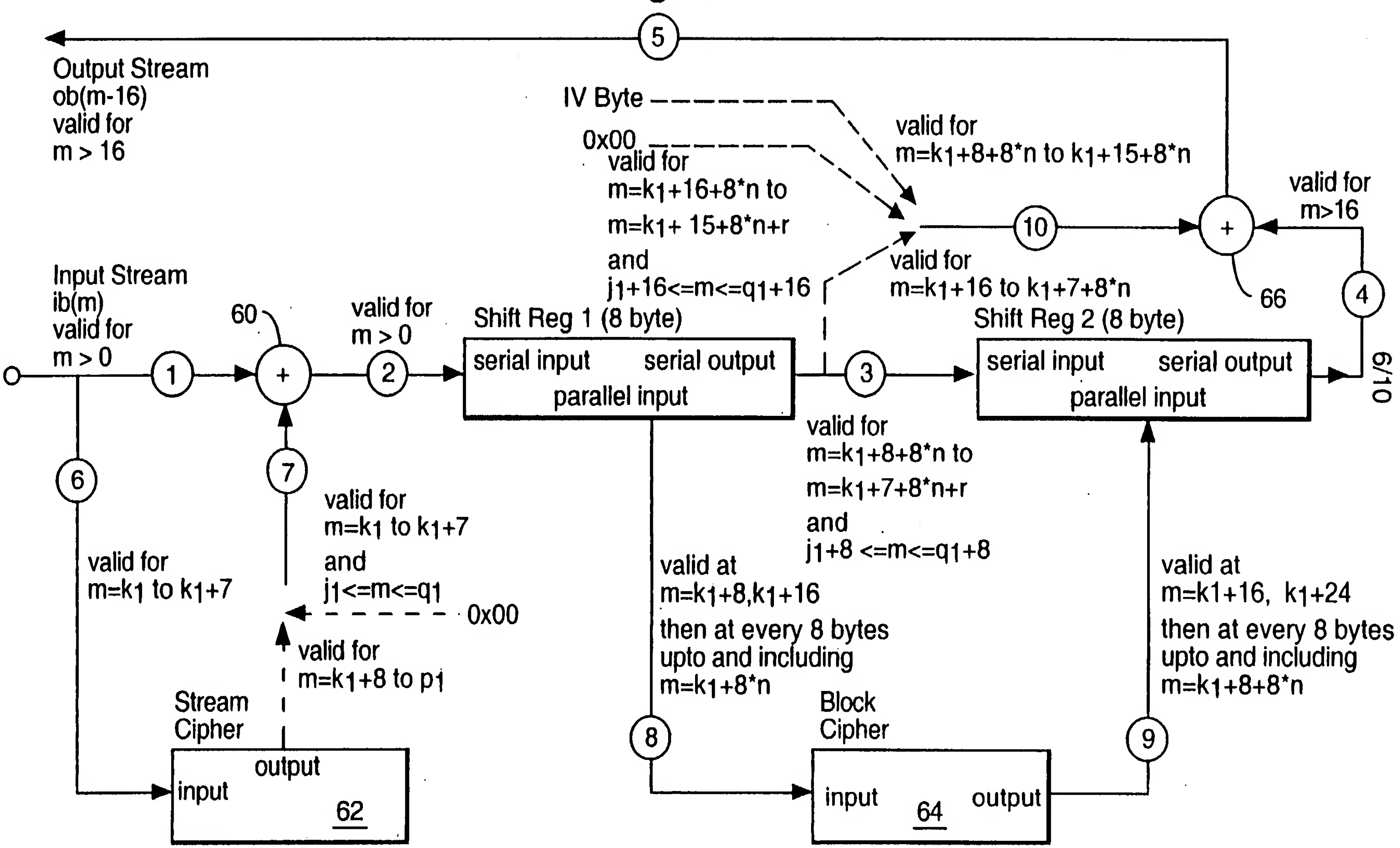

ONE TRANSPORT PACKET
ONE TRANSPORT PACKET
dsc_o_data
SB
PLAIN DATA
DE-SCRAMBLED DATA
SB
PLAIN DATA
DE-SCRAMBLED DATA
PLAIN DATA
dsc_o_pacst
INTERFACE TIMING FOR BOTH INTERFACES
clock
data_val
data
1
2
3
4
5
busyb
captured data
1
2
3
4

Fig.6.



Output Stream
ob(m-16)
valid for
m > 16
IV Byte
0x00
valid for
m=k1+16+8*n to
m=k1+ 15+8*n+r
and
j1+16<=m<=q1+16
valid for
m=k1+8+8*n to k1+15+8*n
valid for
m>16
valid for
m=k1+16 to k1+7+8*n
66
Input Stream
ib(m)
valid for
m > 0
60
valid for
m > 0
Shift Reg 1 (8 byte)
serial input
serial output
parallel input
Shift Reg 2 (8 byte)
serial input
serial output
parallel input
valid for
m=k1+8+8*n to
m=k1+7+8*n+r
and
j1+8 <=m<=q1+8
valid for
m=k1 to k1+7
valid for
m=k1 to k1+7
and
j1<=m<=q1
0x00
valid for
m=k1+8 to p1
valid at
m=k1+8,k1+16
then at every 8 bytes
upto and including
m=k1+8*n
valid at
m=k1+16, k1+24
then at every 8 bytes
upto and including
m=k1+8+8*n
Stream
Cipher
input
output
62
Block
Cipher
input
64
output
1
2
3
4
5
6
7
8
9
10
+
+

Fig.7.

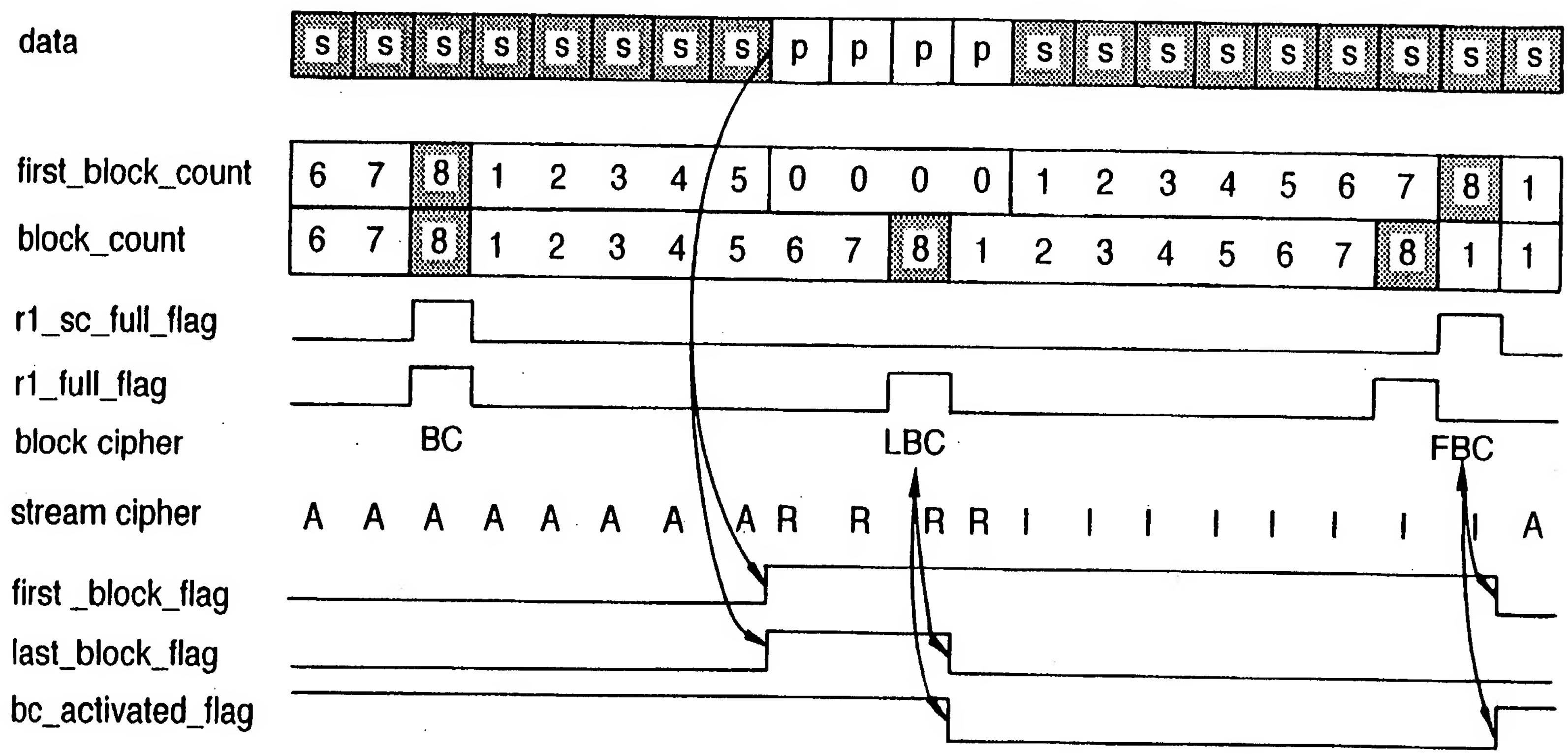

data
first_block_count
block_count
r1_sc_full_flag
r1_full_flag
block cipher
stream cipher
first _block_flag
last_block_flag
bc_activated_flag
s s s s s s s s p p p p s s s s s s s s s
6 7 8 1 2 3 4 5 0 0 0 0 1 2 3 4 5 6 7 8 1
6 7 8 1 2 3 4 5 6 7 8 1 2 3 4 5 6 7 8 1 1
BC
LBC
FBC
A A A A A A A A R R R R I I I I I I I I A

Fig.8.

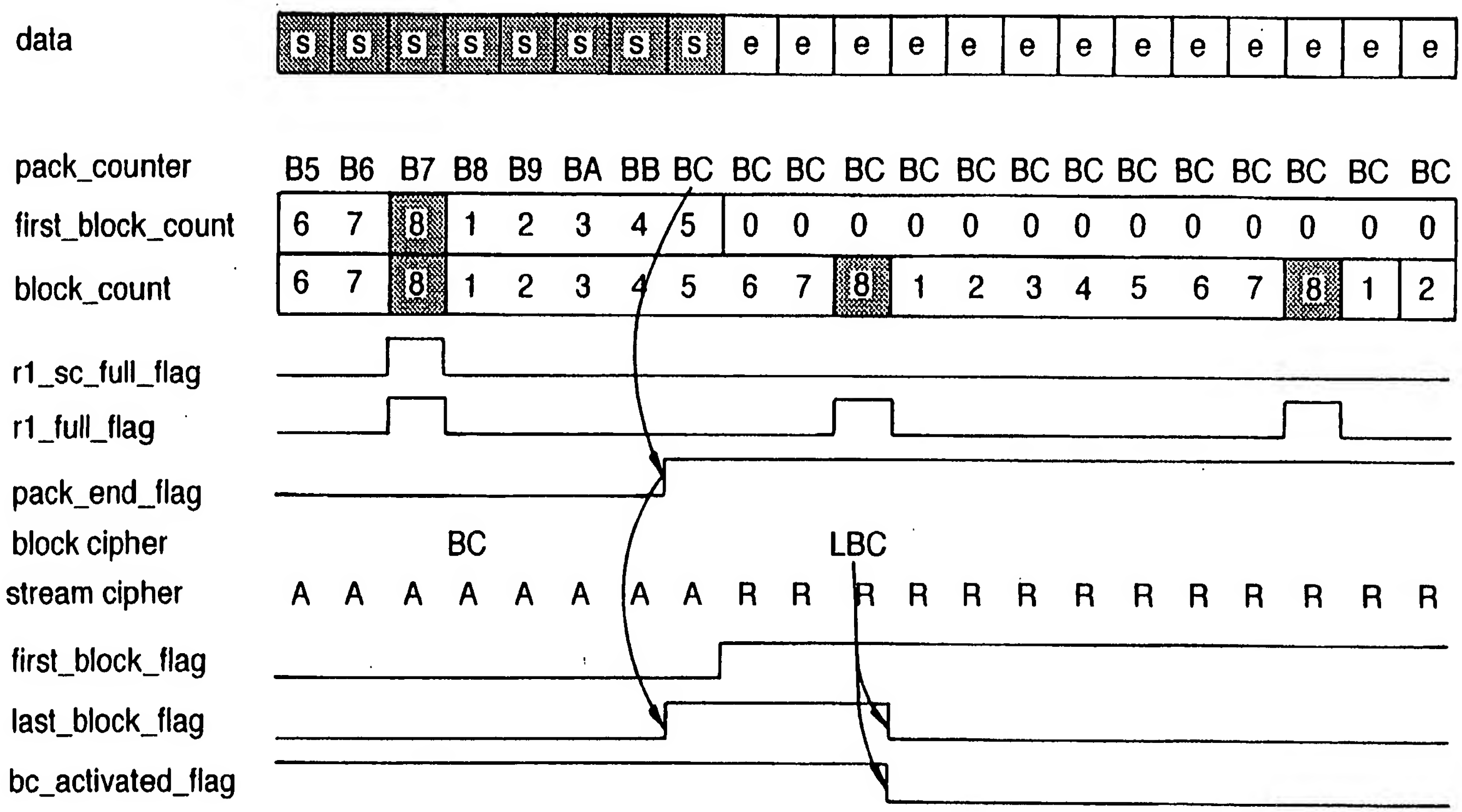

data
S S S S S S S S e e e e e e e e e e e e e
pack_counter
B5 B6 B7 B8 B9 BA BB BC BC BC BC BC BC BC BC BC BC BC BC BC BC
first_block_count
6 7 8 1 2 3 4 5 0 0 0 0 0 0 0 0 0 0 0 0 0
block_count
6 7 8 1 2 3 4 5 6 7 8 1 2 3 4 5 6 7 8 1 2
r1_sc_full_flag
r1_full_flag
pack_end_flag
block cipher
BC
LBC
stream cipher
A A A A A A A A R R R R R R R R R R R R R
first_block_flag
last_block_flag
bc_activated_flag

Fig.9.

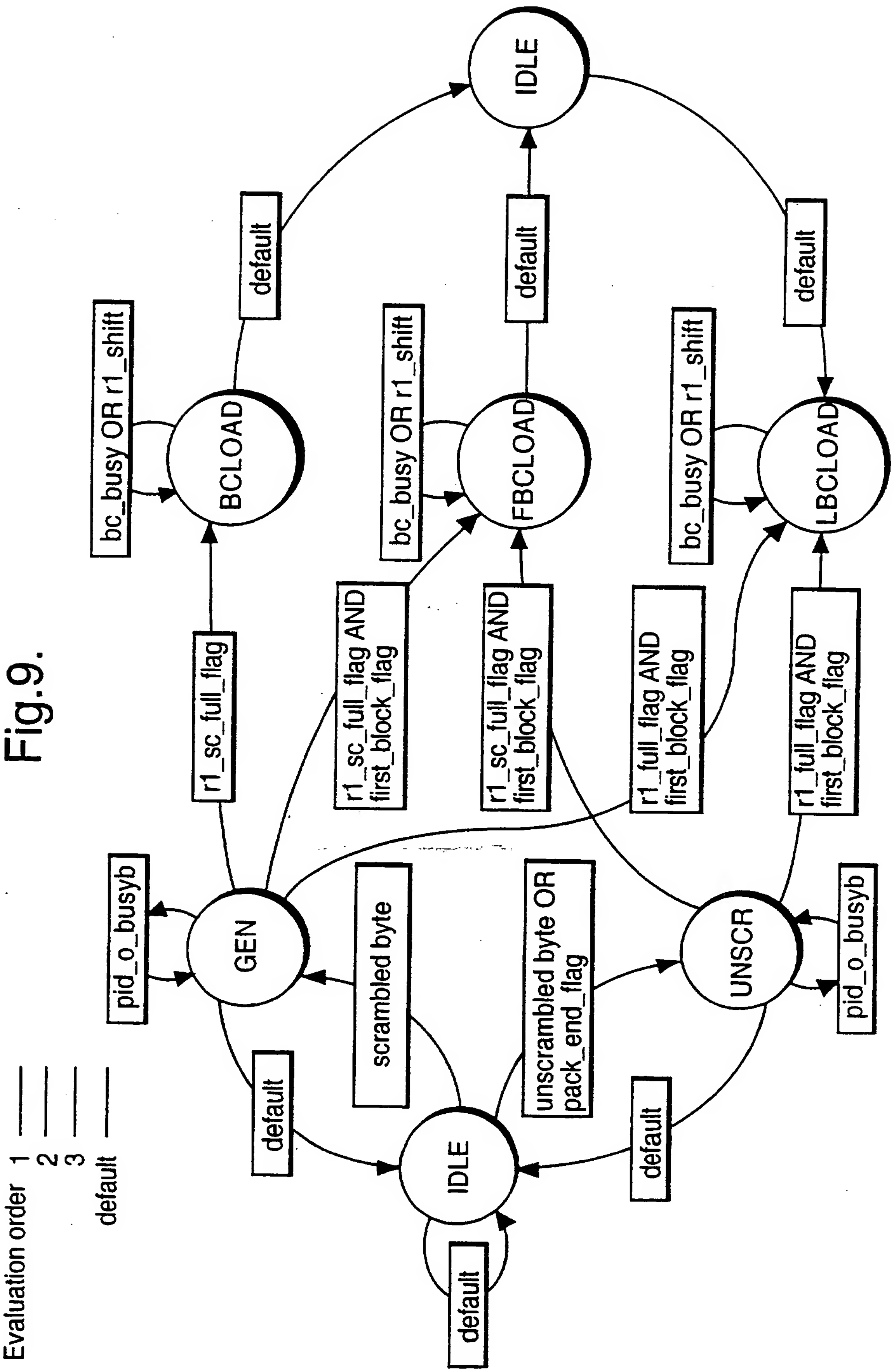
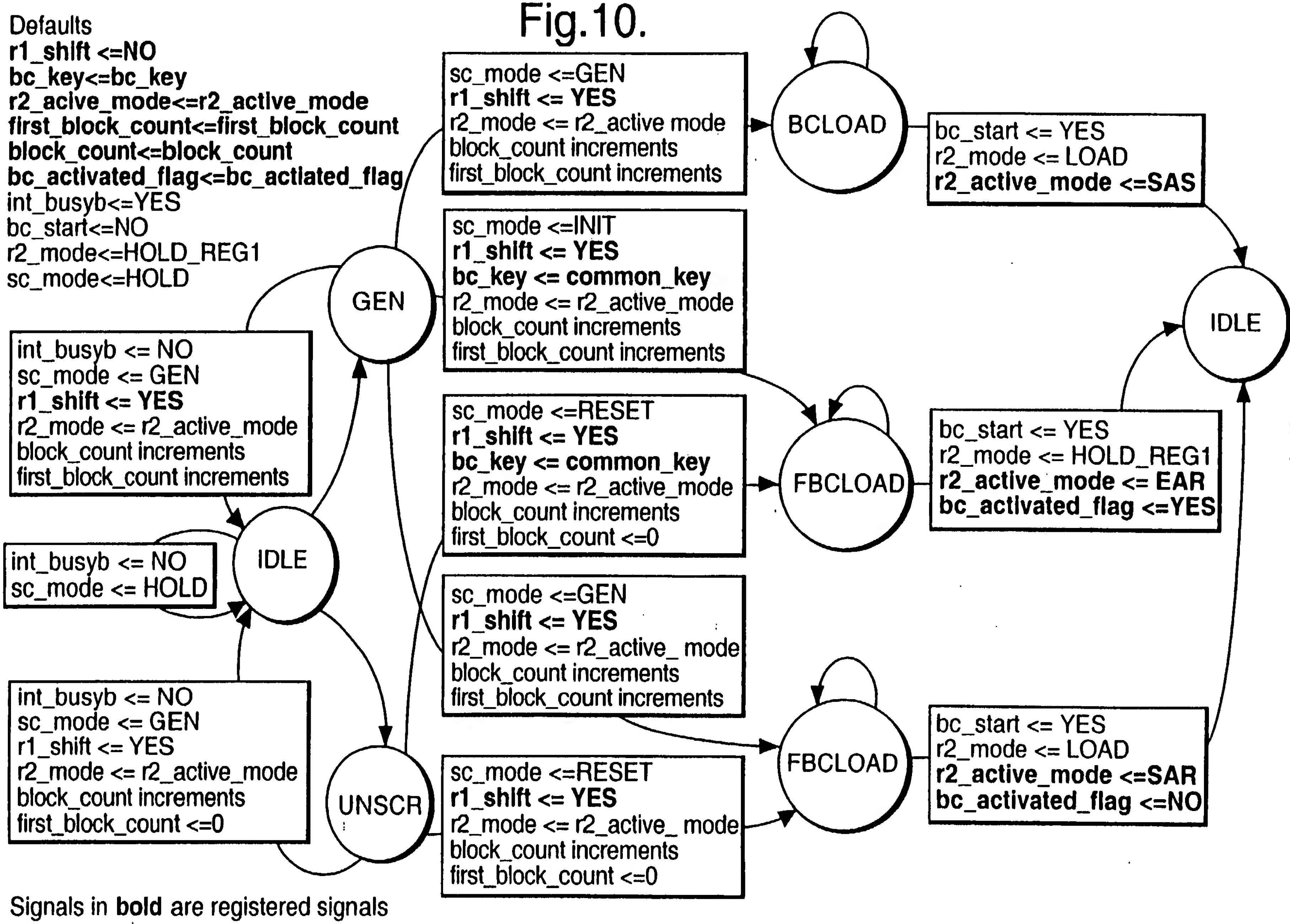

Evaluation order 1
2
3
default
IDLE
default
GEN
pid_o_busyb
default
scrambled byte
unscrambled byte OR pack_end_flag
UNSCR
pid_o_busyb
default
r1_sc_full_flag
r1_sc_full_flag AND first_block_flag
r1_sc_full_flag AND first_block_flag
r1_full_flag AND first_block_flag
r1_full_flag AND first_block_flag
BCLOAD
bc_busy OR r1_shift
FBCLOAD
bc_busy OR r1_shift
LBCLOAD
bc_busy OR r1_shift
default
default
default
IDLE

Fig.10.


Defaults
r1_shift <=NO
bc_key<=bc_key
r2_acive_mode<=r2_active_mode
first_block_count<=first_block_count
block_count<=block_count
bc_activated_flag<=bc_actiated_flag
int_busyb<=YES
bc_start<=NO
r2_mode<=HOLD_REG1
sc_mode<=HOLD
GEN
IDLE
UNSCR
int_busyb <= NO
sc_mode <= GEN
r1_shift <= YES
r2_mode <= r2_active_mode
block_count increments
first_block_count increments
int_busyb <= NO
sc_mode <= HOLD
int_busyb <= NO
sc_mode <= GEN
r1_shift <= YES
r2_mode <= r2_active_mode
block_count increments
first_block_count <=0
sc_mode <=GEN
r1_shift <= YES
r2_mode <= r2_active mode
block_count increments
first_block_count increments
sc_mode <=INIT
r1_shift <= YES
bc_key <= common_key
r2_mode <= r2_active_mode
block_count increments
first_block_count increments
sc_mode <=RESET
r1_shift <= YES
bc_key <= common_key
r2_mode <= r2_active_mode
block_count increments
first_block_count <=0
sc_mode <=GEN
r1_shift <= YES
r2_mode <= r2_active_ mode
block_count increments
first_block_count increments
sc_mode <=RESET
r1_shift <= YES
r2_mode <= r2_active_ mode
block_count increments
first_block_count <=0
BCLOAD
FBCLOAD
FBCLOAD
bc_start <= YES
r2_mode <= LOAD
r2_active_mode <=SAS
bc_start <= YES
r2_mode <= HOLD_REG1
r2_active_mode <= EAR
bc_activated_flag <=YES
bc_start <= YES
r2_mode <= LOAD
r2_active_mode <=SAR
bc_activated_flag <=NO
IDLE
Signals in bold are registered signals

2322994

DIGITAL VIDEO BROADCASTING

The present invention relates to the field of digital video, and in particular to digital video broadcasting.

Scrambling techniques are commonly used in video applications such as payment-on-demand cable TV. A common technique for scrambling analogue video signals is to randomise the position of the horizontal synchronisation pulse. Frequently, an incoming video signal will contain sections of scrambled video interleaved with sections of unscrambled video and it is necessary to distinguish between them in the processing circuits - see for example US-A-4,926,477.

As regards digital broadcasting (DVB) various standards have been set, and in particular a standard has been set, known as the Common scrambling specifications devised by the European project for digital video broadcasting, concerning the scrambling and descrambling of data (the DVB algorithm). Essential characteristics of the DVB algorithm are shown in Figure 1. A packet 2 of data having a header field HEADER followed by n consecutive blocks (of 8 bytes) of scrambled data SB(n) is applied to a stream cipher 4, of a form defined by the common scrambling specifications. The scrambled data is deciphered on a block by block basis with the first block SB(1) being used for initialisation, and each subsequent block being subject to the stream cipher CB and then passed as an intermediate block IB(n) to a block cipher unit 6, also of a form defined by the common scrambling specification. Each block is subject to a block deciphering operation BD and descrambled blocks DB(n) are output for assembly into a descrambled packet 8.

A more specific implementation of the descrambler of Figure 1 is shown in Figure 2, wherein scrambled data is applied to a stream cipher unit 10 and a first input of an exclusive OR gate 12. The output of cipher unit 10 is applied to a second input of gate 12. The output of gate 12 is applied to an 8 byte register 14 (herein referred to as Reg1), which provides an 8 byte delay. A parallel output of Reg1 is coupled to a block cipher unit 16. A serial output of Reg1 is coupled via a two way switch 18 to an 8 byte register 20 (providing a further 8 byte delay- referred to herein as Reg2) and to a first input of an exclusive OR gate 22. The output of block cipher unit 16 is connected as a parallel input to Reg2 and a serial output of Reg2 is applied to a second input of exclusive OR gate 22. The output of gate 22 provides descrambled data. It will be understood that the presence of the two registers Reg1, Reg2 in the data path stream creates substantial 16 byte delay. The notation in Figure 2 is as follows:

k indexes scrambled bytes through the stream

p end of a scrambled field

n number of complete 8 bytes blocks in a scrambled field

r the residue that is left in a scrambled field after the last complete 8 byte block of that scrambled field

For compressed video signals, compressed according to the MPEG standards, the application of the DVB descrambler algorithm to MPEG transport data streams requires that the fields of scrambled data contained in the received transport stream be extracted from the stream, descrambled according to the DVB descrambler algorithm, then reinserted into their original place in the received transport stream.

Where the transport stream contains interleaved sections of scrambled video and unscrambled video (plain text) a problem therefore arises in that the descrambling of the scrambled sections naturally introduces a time delay and it is not therefore possible to reinsert the descrambled sections to their original place in the transport steam without further modification of the receiving system.

It is an object of the invention to provide a simple and inexpensive means of descrambling scrambled data in a transport stream containing sections of both scrambled and unscrambled data.

In accordance with the present invention, there is provided apparatus for processing a stream of digital video broadcast data containing interleaved sections of scrambled data and unscrambled data, the apparatus including a common data flow path provided both for sections of scrambled data and sections of unscrambled data, one or more data flow path loops extending from and back to said common data flow path and containing cipher means to enable the descrambling of scrambled data, and a control means capable of assuming a plurality of control states for selectively controlling the passage of data through the data flow paths to enable passage of unscrambled data sections in said common data flow path and descrambling of scrambled data sections in said data flow loops, while maintaining the relative positions of the data sections in this data stream.

In accordance with the invention, since a single main data flow path is provided, with control of the main data flowpath and the data flow path loops, the passage of unscrambled sections and scrambled sections of data can be regulated to avoid problems of reinsertion of descrambled data and resynchronisation to a single data stream.

The present invention provides in a specific aspect apparatus for processing digital video broadcast data comprising interleaved sections of scrambled and unscrambled data, the apparatus comprising:

a data input terminal coupled to a first input of an exclusive OR gate means, a stream cipher means having an input connected to said data input terminal and an output connected to a second input of the exclusive OR gate means;

a first shift register means having an input coupled to receive the output of the exclusive OR gate means, and having first and second outputs, a second shift register means having a first input coupled to the first output of the first shift register means, and having a second input and an output, a block cipher means coupled between the second

output of the first shift register means and the second input of the second shift register means; and

a control means responsive to whether data at said data input terminal is scrambled data or unscrambled data for selectively enabling said stream cipher means, block cipher means and first and second shift register means (a) to pass an unscrambled data directly through said first and second shift register means and (b) to pass scrambled data through said stream and block cipher means.

In accordance with the invention, since a single data flow path is provided a more secure, reliable and inexpensive system is provided. Whilst other arrangements may be envisaged for example splitting the data into completely separate data flow paths for scrambled and unscrambled data with appropriate delays in the flow paths to maintain correct timing relationships, this would result in a more expensive system.

The single data stream path in accordance with the invention is preferably arranged that it cannot pass both a scrambled byte of data and a non-scrambled data and that further the two shift register means do not contain gaps between successive fields of data. These two conditions mean that scrambled and non-scrambled fields follow each other without either a gap or an overlap, in data (but not necessarily in time intervals), and since the non-scrambled data uses the same shift register means, providing a 16 byte time delay as is used by the scrambled data, the problem of inserting non-scrambled data into the descrambled data stream does not arise and is automatically solved.

The control means preferably comprises a control state machine, preferably occupying a number of predefined states in which it issues appropriate control signals to control the data stream. As preferred a key state machine is provided for controlling the issuing of keys to the cipher means for the descrambling process. A packet counter is preferably provided to count the number of bytes of the current signal packet of the incoming data stream up to a maximum of 184 bytes as permitted by MPEG-2. A first block counter is provided to count the number of bytes of a new scrambled field modulo 8; this counter controls said first shift register means. As preferred a second block counter is provided to count the bytes of a second block of data of a new scrambled field

modulo 8. Two block counters are needed since there may be less than 8 bytes of unscrambled data between two successive scrambled fields.

A preferred embodiment of the invention will now be described with reference to the accompanying drawings wherein:-

Figure 1 is a diagram illustrating the concept of descrambling, according to the ETSI Common scrambling specifications;

Figure 2 is a more detailed implementation of a descrambling mechanism according to the ETSI Common scrambling specifications;

Figure 3 is a block diagram of a preferred embodiment of descrambling apparatus according to the present invention;

Figures 4 and 5 are wave form diagrams for apparatus of Figure 3;

Figure 6 is a more detailed block diagram of the preferred embodiment of descrambling apparatus according to the present invention;

Figures 7 and 8 are timing diagrams showing operation of counters and flags in the apparatus of Figure 3;

Figures 9 and 10 are diagrams indicating the operation of the control state machine in terms of transitions between states (9) and actions taken in the various states (10); and

Figure 11 is a schematic view of an integrated circuit chip incorporating the circuit of Figure 3.

Referring to Figure 3, the descrambling apparatus according to the present invention includes a demultiplexing function 30 and a descrambler function 32, both indicated in dotted lines. Function 30 includes a packet identification processor 34. Function 32 includes a key control and key generator unit 36, a control state machine 38,

and a descrambler unit 40. Machine 38 includes counters 42, 44, 46 as will be described below.

Referring to Figures 4 and 5 the format of transport packets are shown as preceded by a synchronisation byte SB and include 188 bytes of data which may be scrambled data, plain data or a mixture of the two. Where scrambled data is provided, preceding instruction bytes IB may also be provided. The format of an instruction byte IB is shown in Figure 4 as comprising bits 0 to 3 as a key index and bits 4 and 5 as SC control bits. Packet identification unit 34 provides the incoming packets, on pid_i_data to machine 38, and produces the wave forms shown in Figure 4, pid_i_decidb, pid_i_pacst and pid_i_sr, which provide waveform steps in response to instruction bytes, synchronisation bytes and scrambled data sections respectively to machine 38.

As indicated in Figure 5, descrambler unit 40 provides descrambled data assembled in packets dsc_o_data, together with a signal, dc_o_pacst in response to synchronisation bytes at the start of packets. Timing and validation signals clock, data_val, data, busyb and captured data are provided as indicated.

The control state machine 38 includes a counter 42, pack_counter for counting the bytes in a packet, a counter 44 for counting blocks of 8 bytes, first_block_count and a second counter 46 for counting bytes of blocks, block_count. The two block counters count the number of bytes of a new scrambled field modulo 8 to generate blocks. When the counters reach 8 they wrap round to 1. A zero value means the byte currently is unscrambled. Two block counters are needed since there may be less than 8 bytes of unscrambled data between two successive scrambled fields. The counters produce various flags as indicated in Figures 7 and 8 for controlling operation of the descrambler as referred to below.

Referring now to Figure 6, this shows a more detailed diagram of unit 40 of Figure 3, an input stream ib(m) (in Figure 6, m indexes the incoming bytes, k indexes scrambled bytes and j to q refer to unscrambled bytes) is input on a data flow path (1) to an exclusive OR gate 60. The input stream is also applied on a data path (6) to a stream cipher unit 62, whose output is coupled by a data flow path (7) to a second input of

exclusive OR gate 60. The output of exclusive OR gate 60 is coupled on data flow path (2) to a serial input of first shift register Reg1. Reg1 is an 8 byte shift register and provides a parallel output on a data flow path (8) and a serial output on a data flow path (3). Data flow path (3) is coupled to the serial input of 8 byte shift register Reg2 Data flow path (8) is coupled to a block cipher unit 64, and the output of block cipher unit 64 is coupled on a data flow path (9) to the parallel input of Reg2. The serial output of Reg2 is coupled via a data flow path (4) to an input of an exclusive OR gate 66, the output of which provides a descrambled output on data flow path (5). A further data flow path (10) is connected from the output of Reg 1 to the second input of gate 66, in order to complete the DVB descrambling algorithm.

In use, scrambled data, m=k, is directed through the two signal path loops (6,7), (8,9) including ciphers 62, 64, and along signal path 10 to provide descrambled data. The two shift registers Reg1 and Reg2 , supplying data to and receiving data from block cipher 64 eight bytes at a time, introduce a 16 byte delay. Sections of non-scrambled data, j <= m <=q are directed along the common data flow path 1, 2, 3, 4, 5. Since the common signal flow path includes the two byte serial shift registers Reg1 and Reg2, there is automatically provided the 16 byte time delay compensation to maintain timing with scrambled data sections, without the need for external delay elements.

The data flow paths 1, 2, 3, 4 and 5 are specified as indicated by the values of m in Figure 6 to achieve two aims. Firstly that they never simultaneously pass both a scrambled byte and a non-scrambled byte, and secondly that they never result in Reg1 or Reg2 containing gaps between successive fields of data. These two conditions mean the scrambled and non-scrambled fields follow each other without either a gap or an overlap, and since the non-scrambled data uses the same circuitry to provide the 16 byte time delay as is used by the scrambled data, the problem of inserting the non-scrambled data back into the descrambled data stream does not arise and is therefore automatically solved.

Referring now back to Figure 3 together with the wave form diagrams of Figures 4 to 8 and the control operation diagrams of Figures 9 and 10, the function of the unit

shown in Figure 3 and 6 are as follows:

packet counter 42: this counts the number of bytes since the last packet start. This counter is used to determine when the end of a packet is reached (188 bytes).

first block counter 44: this counts the bytes from the beginning of a new scrambled field modulo 8. This means that when the count reaches 8 it wraps around to 1. From this it is possible to determine when Reg1 is full with a complete scrambled block from a new scrambled field. The counter is defined from 0 to 8 with the 0 having the meaning that the byte currently in the first stage of Reg1 is unscrambled. Therefore unscrambled fields are not counted by this counter.

second block counter 46: this counts the bytes from the beginning of the second block of a new scrambled field modulo 8. From this it is possible to determine when Reg1 is full with the next block of scrambled data from the stream cipher.

Two block counters are needed since there may be less than 8 bytes of unscrambled data between two successive scrambled fields. In such a case Reg1 will not fill with the unscrambled data before the next scrambled field, i.e. Reg1 will contain both the end of the last scrambled field and the beginning of the next scrambled field. The control state machine needs to know when the last block is ready for emptying from the block cipher and when the first block has reached the end of Reg1, so two counters are needed.

From these counters a series of flags are produced as follows as shown in Figures 7 and 8:-

first_block_flag: Indicates the block in Reg1 is the first block of a scrambled field

last_block_flag: Indicates the block in the block cipher is the last block of a scrambled field.

r1_full_flag: Indicates that Reg1 is full with 8 new bytes (one block) of data. It is set when block count is 7.

r1_sc_full_flag: Indicates that Reg1 is full with 8 new scrambled bytes of data. It is set when first block counter is 7.

pack_end_flag : Indicates that 188 bytes have been loaded since the last packet start.

Control State Machine 38: The state machine is held in signal 'current state' with the value of the state after the next clock being held in Register 'next state'. This state machine has 6 states - IDLE, GEN, UNSCR, BCLOAD, FBCLOAD, LBCLOAD as indicated in Figures 9 and 10. Referring to Figures 9 and 10, these states are as follows:-

IDLE

Default state after reset. The internal controls in the descrambler are such that a new byte can be accepted. If a new byte is available and it is scrambled then the next state is GEN. If a new byte is available and it is un-scrambled then the next state is UNSCR. Otherwise the next State is IDLE.

GEN

The processing state for scrambled data. The stream cipher requires two clock cycles to process each byte. By default this will occur by advancing IDLE to GEN then back to IDLE. This is the normal flow for a scrambled data byte except in the following cases: if Reg1 is full and the block cipher contains the last block of a scrambled field then the next state is LBCLOAD; if Reg1 is full a block from a scrambled field and it is neither the first nor last block of that field then the next state is BCLOAD, if none of these cases are true then the next state is IDLE.

UNSCR

The processing state for unscrambled data. In reality unscrambled data only needs one cycle to process but providing a special state has two advantages. Firstly the cycle behaviour is the same for scrambled and unscrambled bytes. This inevitably reduces the special case requirements in the code thus increasing its reliability. Secondly the latency of the descrambler is the same for scrambled and unscrambled data which can simplify external interfacing requirements in some systems. In this state if Reg1 is full and the last block of a scrambled field is in the block cipher then the next state is LBCLOAD,

and if Reg1 is full with the first block of a scrambled field then the next state is FBCLOAD; Otherwise the next state is IDLE.

BCLOAD

The processing state for loading and unloading the block cipher. This state is entered from GEN when Reg1 contains a new scrambled block that is neither the first block nor the scrambled residue of a scrambled field. This state waits until two conditions are true: firstly we have completed the shifting in of data; secondly the block cipher is currently idle i.e. it has finished its decipherment of the previous block. When these are true: Reg2 is loaded with the contents of the block cipher; the block cipher is loaded with the contents of Reg1 and the block cipher is started; Reg2 is set to mode 'Shift As Scrambled' (SAS); the next state is IDLE.

FBCLOAD

The processing state for loading the block cipher with the first block of a scrambled field. This state is entered from GEN when Reg1 contains a new scrambled block that is also the first block of a new scrambled field. In this case the block cipher does not contain a block from the current field, Any last block from the previous field will have been emptied into Reg2 by a LBCLOAD state. To correctly operate this scheme it is necessary to have at least one byte of unscrambled data between successive scrambled fields. This state waits for the same two conditions to be true as BCLOAD then performs the following: the block cipher is loaded with the contents of Reg1 and the block cipher is started; Reg2 is set to mode 'Empty As Residue' (EAR); the block cipher key Register 'bc key' is loaded with the latest 'common key'; the next state is IDLE.

LBCLOAD

The processing state for handling any possible scrambled residue for a scrambled field. This state is entered from the IDLE state when Reg1 is detected as having 8 new bytes that have not yet been processed and not all of those bytes are from the current scrambled field. Scrambled residue does not pass through the block cipher, but is instead serially passed into Reg2. There will be the previous block of data in the block

cipher. This state waits until two conditions are true: firstly we have completed the shifting in of data; secondly the block cipher is currently idle i.e. it has finished its decipherment of the previous block. When these are true; Reg2 is loaded with the contents of the block cipher; Reg2 is set to mode 'Shift As Reside' (SAR); the next state is IDLE.

Key State Machine 36: The descrambling process requires the use of a common key, there being a possible large number of such keys. An instruction byte IB (Figure 4) passed in the data stream contains the key index information necessary to determine - which of those keys is to be used. When the instruction byte appears in the data stream then the key state machine takes control of the input interface and performs the key look up operation.

The key state machine is held in signal 'key state' and the next state of the machine is held in state 'next key state'. The default state is IDLE. If the current state is IDLE and an instruction byte arrives (this is indicated by the PID-processor 34) then the next state progression is KEY 1 followed by KEY 2 followed by IDLE. During KEY 1 and KEY 2 the signal 'look up key' is asserted '1' causing a look up of the key addressed by two Register values. These are 'sc bits' and 'key index' which are constructed from the instruction byte. The key file returns the key value on the bus COMMON_KEY. This is registered inside the key file in descrambler 40 and control state machine 38

CONTROL STATE MACHINE 38:

As regards the specific construction of the control state machine, it will be understood that it is practice within the art to define a machine construction in terms of a software routine written in a hardware programming language, VHDL, and for a computer directly to translate such routines into a set of layout diagrams and chip masks for a chip consisting of hard-wired logic gates, the chip being indicated schematically in Figure 11 There is not normally generated anything which corresponds to a traditional functional block diagram.

Accordingly the construction of the control state machine is defined by the following routine:

```
Process         : next_state_proc
This process calculates the next state of the control state machine.  It also flags
when to start the block cipher, when to load Reg2 and which mode to set for
Reg2.
next_state_proc  :  process
    (current_state, new_byte_flag, captured, bc_busy,
    first_block_flag, r1_sc_full_flag, int_r1_shift,
    r1_full_flag, reg2_active_mode, last_block_flag,
    int_bc_key, common_key, pack_end_flag, if_empty)
begin
    a_bc_key <= int_bc_key;
    bc_start    <= NO;
    reg2_load_flag <= NO;
    a_reg2_active_mode <= reg2_active_mode;
    exit_state <= NO;
    case (current_state) is
    when IDLE =>
        if (new_byte_flag = YES and captured.sc = SCRAMBLED ) then
         next_state <= GEN;
        elsif ( new_byte_flag = YES and captured.sc = SCRAMBLED ) then
  next_state <= UNSCR;
elsif ( pack_end_flag = YES ) then
         next_state <= UNSCR;
        else
         next_state <= IDLE;
        end if;
when GEN =>
     if (if _empty = NO ) then
         next_state <= current_state;
     elsif (rl_full_flag = YES and last_block_flag = YES ) then
         next_state <= LBCLOAD;
         exit_state <= YES;
     elsif (r1_sc_full_flag = YES ) then
        if (first_block_flag = YES ) then
             next_state <= FBCLOAD;
             a_bc_key <= common_key;
             exit_state <= YES;
        else
             next_state <= BCLOAD;
             exit_state <= YES;
        end if;
    else
        next_state <- IDLE;
        exit_state <- YES;
```

```
        end if ;
    when USCR =>
        if (if_empty = NO ) then
            next_state <= current_state;
        elsif (r1_full_flag = YES and last_block_flag = YES ) then
            next_state <= LBCLOAD;
            exit_state <= YES;
        elsif (r1_sc_full_flag = YES and first_block_flag = YES ) then
            next_state <= FBCLOAD;
            a_bc_key <= common_key;
            exit_state <= YES;
        else
            next_state <= IDLE;
            exit_state <= YES;
        end if;
    when LBCLOAD =>
        if (int_r1_shift = YES or bc_busy = YES ) then
            next_state <= current_state;
        else
            next_state <= IDLE;
            reg2_load_flag <= YES;
            a_reg2_active_mode <= SAR;
        end if;
    when FBCLOAD =>
        if ( int_r1_shift = YES or bc_busy = YES ) then
            next_state <= current_state;
        else
                    next_state <= IDLE
            bc_start <= YES;
            a_reg2_active_mode <= EAR;
        end if:
    when BCLOAD =>
        if (int_r1_shift = YES or bc_busy = YES ) then
            next_state <= current_state;
        else
            next_state <= IDLE;
            bc_start <= YES;
            reg2_load_flag <= YES;
            a_reg2_active_mode <=SAS;
        end if;
    when others =>
        next_state <= IDLE;
    end case;
end process next_state_proc;
```

It will be understood that control state machine 38 will be essentially the same construction, even if expressed differently using different nomenclature, changes in algorithm producing an equivalent result, and expressed in a different programming language, all of which changes will be apparent to the person skilled in the art. It is to be understood that the claims appended hereto are intended to cover all such variations.

CLAIMS

1. Apparatus for processing a stream of digital video broadcast data containing interleaved sections of scrambled data and unscrambled data, the apparatus including a common data flow path provided both for sections of scrambled data and sections of unscrambled data, one or more data flow path loops extending from and back to said common data flow path and containing cipher means to enable the descrambling of scrambled data, and a control means capable of assuming a plurality of control states for selectively controlling the passage of data through the data flow paths to enable passage of unscrambled data sections in said common data flow path and descrambling of scrambled data sections in said data flow loops, while maintaining the relative positions of the data sections in the data stream.

2. Apparatus according to claim 1, including a first signal path loop including a stream cipher means, and a second signal path loop including a block cipher means and including a first shift register (Reg1) in the common data flow path at the input to the second loop and a second shift register (Reg2) in the common data flow path at the output of the second loop.

3. Apparatus according to claim 1, including a packet identification processing means for identifying receipt of a packet of video data and for providing control signals indicating the presence of scrambled and unscrambled data to said control means.

4. Apparatus according to claim 1, including a key state means for generating key signals for use by said cipher means in response to instruction bytes in an incoming pocket of data.

5. Apparatus for processing digital video broadcast data comprising interleaved sections of scrambled and unscrambled data, the apparatus comprising:

a data input terminal coupled to a first input of an exclusive OR gate means, a stream cipher means having an input connected to said data input terminal and an output connected to a second input of the exclusive OR gate means;

a first shift register means having an input coupled to receive the output of the exclusive OR gate means, and having first and second outputs, a second shift register means having a first input coupled to the first output of the first shift register means, and having a second input and an output, a block cipher means coupled between the second output of the first shift register means and the second input of the second shift register means; and

a control means responsive to whether data at said data input terminal is scrambled data or unscrambled data for selectively enabling said stream cipher means, block cipher means and first and second shift register means (a) to pass an unscrambled data directly through said first and second shift register means and (b) to pass scrambled data data through said stream and block cipher means.

6 Apparatus according claim 5, including first and second block counters (84, 86) for counting scrambled field data.

7. Apparatus according to claim 6, wherein a first block counter counts from the beginning of a new scrambled field and a second block counter counts from the beginning of a second block a new scrambled field.

8. Apparatus according to claim 5, including a packet counter for counting the number of bytes since the start of the packet.

9. Apparatus according to claim 2, wherein said control state machine can occupy the following states:

An IDLE state, a processing state for scrambled data GEN, a processing state for unscrambled data UNSCR, a processing state for loading and unloading a block cipher BCLOAD, a processing state for loading the block cipher with the first block of a scrambled field FBCLOAD, and a processing state for handling any scrambled residue LBCLOAD.

10. Apparatus according to claim 9, wherein the construction of the control state machine is defined by the result of the following process:

```
next_state_proc  :  process
        (current_state, new_byte_flag, captured, bc_busy,
        first_block_flag, r1_sc_full_flag, int_r1_shift,
        r1_full_flag, reg2_active_mode, last_block_flag,
        int_bc_key, common_key, pack_end_flag, if_empty)
    begin
        a_bc_key <= int_bc_key;
        bc_start    <= NO;
        reg2_load_flag <= NO;
        a_reg2_active_mode <= reg2_active_mode;
        exit_state <= NO;
        case (current_state) is
        when IDLE =>
            if (new_byte_flag = YES and captured.sc = SCRAMBLED ) then
             next_state <= GEN;
            elsif ( new_byte_flag = YES and captured.sc = SCRAMBLED ) then
      next_state <= UNSCR;
    elsif ( pack_end_flag = YES ) then
             next_state <= UNSCR;
            else
     next_state <= IDLE;
            end if;
    when GEN =>
        if (if _empty = NO ) then
            next_state <= current_state;
        elsif (r1_full_flag = YES and last_block_flag = YES ) then
    next_state <= LBCLOAD;
            exit_state <= YES;
    elsif (r1_sc_full_flag = YES ) then
            if (first_block_flag = YES ) then
```

```
            next_state <= FBCLOAD;
            a_bc_key <= common_key;
            exit_state <= YES;
        else
            next_state <= BCLOAD; .
            exit_state <= YES;
        end if;
    else
        next_state <- IDLE;
        exit_state <- YES;
    end if ;
  when USCR =>
      if (if_empty = NO ) then
         next_state <= current_state;
      elsif (r1_full_flag = YES and last_block_flag = YES ) then
         next_state <= LBCLOAD;
         exit_state <= YES;
      elsif (r1_sc_full_flag = YES and first_block_flag = YES ) then
         next_state <= FBCLOAD;
         a_bc_key <= common_key;
         exit_state <= YES;
      else
         next_state <= IDLE;
         exit_state <= YES;
      end if;
  when LBCLOAD =>
      if (int_r1_shift = YES or bc_busy = YES ) then
         next_state <= current_state;
      else
         next_state <= IDLE;
         reg2_load_flag <= YES;
         a_reg2_active_mode <= SAR;
      end if;
  when FBCLOAD =>
      if ( int_r1_shift = YES or bc_busy = YES ) then
         next_state <= current_state;
      else
                next_state <= IDLE
         bc_start <= YES;
         a_reg2_active_mode <= EAR;
end if:
  when BCLOAD =>
      if (int_r1_shift = YES or bc_busy = YES ) then
         next_state <= current_state;
      else
         next_state <= IDLE;
         bc_start <= YES;
         reg2_load_flag <= YES;
```

```
        a_reg2_active_mode <=SAS;
      end if;
   when others =>
      next_state <= IDLE;
   end case;
end process next_state_proc;
```

11. An integrated circuit chip incorporating apparatus according to claim 1.

12. A method of processing a stream of digital video broadcast data comprising interleaved sections of scrambled data and unscrambled data, the method comprising providing a common data flow path both for sections of scrambled data and sections of unscrambled data and providing one or more data flow path loops extending from and back to said common data flow path and containing cipher means to enable the descrambling of scrambled data, and selectively controlling the passage of data through the data flow paths to pass unscrambled data sections through said common data flow path and to pass scrambled data sections through said data flow path loops to the scrambled data sections, while maintaining the relative positions of the scrambled and unscrambled data sections in the data stream.

13. A method according to claim 12, including providing a first signal path loop including a stream cipher means, and providing a second signal path loop including a block cipher means and including a first shift register (Reg1) in the common data flow path at the input to the second loop and a second shift register (Reg2) in the common data flow path at the output of the second loop.

14. A method according to claim 12, including identifying receipt of a packet of video data and identifying the presence of scrambled and unscrambled data therein for control of the data flow paths.

15. A method according to claim 12, including generating key signals for use by said cipher means when descrambling data.

16. A method according to claim 13, wherein said processing is divided into

An IDLE state, a processing state for scrambled data GEN, a processing state for unscrambled data UNSCR, a processing state for loading and unloading a block cipher BCLOAD, a processing state for loading the block cipher with the first block of a scrambled field FBCLOAD, and a processing state for handling any scrambled residue LBCLOAD.

17. A method according to claim 16, wherein the processing states are defined as follows:

```
next_state_proc  :  process
        (current_state, new_byte_flag, captured, bc_busy,
        first_block_flag, r1_sc_full_flag, int_r1_shift,
        r1_full_flag, reg2_active_mode, last_block_flag,
        int_bc_key, common_key, pack_end_flag, if_empty)
    begin
        a_bc_key <= int_bc_key;
        bc_start    <= NO;
        reg2_load_flag <= NO;
        a_reg2_active_mode <= reg2_active_mode;
        exit_state <= NO;
        case (current_state) is
        when IDLE =>
            if (new_byte_flag = YES and captured.sc = SCRAMBLED ) then
             next_state <= GEN;
            elsif ( new_byte_flag = YES and captured.sc = SCRAMBLED ) then
      next_state <= UNSCR;
     elsif ( pack_end_flag = YES ) then
             next_state <= UNSCR;
            else
     next_state <= IDLE;
            end if;
     when GEN =>
         if (if _empty = NO ) then
             next_state <= current_state;
         elsif (rl_full_flag = YES and last_block_flag = YES ) then
```

```
next_state <= LBCLOAD;
        exit_state <= YES;
elsif (r1_sc_full_flag = YES ) then
        if (first_block_flag = YES ) then
            next_state <= FBCLOAD;
            a_bc_key <= common_key;
            exit_state <= YES;
        else
            next_state <= BCLOAD;
            exit_state <= YES;
        end if;
    else
        next_state <- IDLE;
        exit_state <- YES;
   end if ;
when USCR =>
     if (if_empty = NO ) then
        next_state <= current_state;
     elsif (r1_full_flag = YES and last_block_flag = YES ) then
        next_state <= LBCLOAD;
        exit_state <= YES;
     elsif (r1_sc_full_flag = YES and first_block_flag = YES ) then
        next_state <= FBCLOAD;
        a_bc_key <= common_key;
        exit_state <= YES;
     else
        next_state <= IDLE;
        exit_state <= YES;
     end if;
when LBCLOAD =>
     if (int_r1_shift = YES or bc_busy = YES ) then
        next_state <= current_state;
     else
        next_state <= IDLE;
        reg2_load_flag <= YES;
        a_reg2_active_mode <= SAR;
    end if;
when FBCLOAD =>
     if ( int_r1_shift = YES or bc_busy = YES ) then
        next_state <= current_state;
     else
               next_state <= IDLE
        bc_start <= YES;
        a_reg2_active_mode <= EAR;
end if:
when BCLOAD =>
     if (int_r1_shift = YES or bc_busy = YES ) then
        next_state <= current_state;
```

```
            else
               next_state <= IDLE;
               bc_start <= YES;
               reg2_load_flag <= YES;
               a_reg2_active_mode <=SAS;
            end if;
       when others =>
            next_state <= IDLE;
       end case;
end process next_state_proc.
```

The Patent Office

23

Application No: GB 9804913.3
Claims searched: All

Examiner: Joe McCann
Date of search: 3 July 1998

Patents Act 1977
Search Report under Section 17

Databases searched:

UK Patent Office collections, including GB, EP, WO & US patent specifications, in:

UK Cl (Ed.P): H4F(FDE)

Int Cl (Ed.6): H04N(7/16,7/167)

Other: Online: WPI, INSPEC

Documents considered to be relevant:

Category	Identity of document and relevant passage	Relevant to claims
A	EP 0805599A2 (OKI)	
X	Prior art acknowledged on pages 1 and 2 and in figure 2 of the application.	1,5,6,12

X	Document indicating lack of novelty or inventive step	A	Document indicating technological background and/or state of the art.
Y	Document indicating lack of inventive step if combined with one or more other documents of same category.	P	Document published on or after the declared priority date but before the filing date of this invention.
&	Member of the same patent family	E	Patent document published on or after, but with priority date earlier than, the filing date of this application.